CMPE 230
System Programming
Project 3 - Minesweeper

Efe Feyzi Mantaroğlu 2021400027

Volkan Bora Seki 2021400156

1 - Introduction

A digital version of the popular Minesweeper game is the objective of the Minesweeper project. A grid of squares is displayed throughout the game, some of which have been planted mines. The goal is to remove all of the mines from the grid without setting any off. Also, there is an extra hint function which shows a grid which is certainly not planted by the view of the player if there exists such a grid. This project is an adaptation of minesweeper to C++ and QT.

2 - Base Game

Base game is implemented using several layouts. Main window is separated into two parts using Vertical Layout and it contains one Horizontal Layout and one Grid Layout. Horizontal Layout contains score label, restart and hint buttons.

Score label is a simple extension of QLabel. It holds the score and increments it by 1 every time when a signal comes. Restart and hint buttons are regular QPushButtons.

Grid Layout contains multiple MineButton objects in a grid. MineButton is an extension of QPushButton that holds its coordinates, flagged and opened status. These MineButtons send signals to gamehandler object. Gamehandler controls the main game according to these signals. It also has a reference to all MineButtons to ease access time. It has a Boolean variable named canPlay that indicate if these objects can be called.

At start, gamehandler generates a map using the N,M,K variables. It tries to put mine to an empty coordinate until placed mines are equal to K. These mines are stored in variable map that is a vector consisting of bool vectors. Then it continues according to incoming signals.

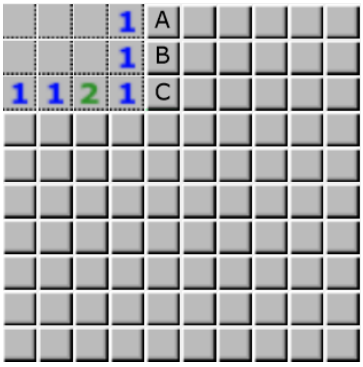
If there is a left button click signal from a MineButton, it retrieves caller's coordinates. It checks if the cell is opened, if the cell is opened, it does not do any action. Then checks if the coordinate is in the map. If it is in the map, player pressed a bomb, so we change canPlay variable to false and show a window indicating the loss. Else, we send score update signal to score label and search its neighbors. By checking every neighbor, we get the neighboring bomb count. We change this MineButton's image according to this number. Also, if this number is zero, we call the same function recursively on neighboring cells leading to depth first search.

Gamehandler keeps track of opened cell count. If sum of this count and the bomb count equals the rows * cols we show a window indicating win and make canPlay status false.

We have modified MineButton to send right click signals. When this signal comes, we first check if the button is opened. If this is true, we do nothing. Else we invert its flagged status and change its image accordingly.

3 - Hint System

This is the most significant part of our code and main feature of the project. We try to find safe cells by creating equations that involve unopened cells that are adjacent to opened cells. We then do gaussian elimination on these equations and find safe cells according to these equations.



Example Grid

We can convert this grid to a matrix. The numbers on the buttons imply neighboring bomb count. So, we can say that adding all neighboring cells must equal the number of the cell. If we take upper-right 1 as an example.

Since cell has number 1, the equation is	A + B = 1,
if we take the cell underneath it, we get the equation	A + B + C = 1
we can subtract upper equation from the lower one, we get :	C = 0

so, we can safely say that C block is safe to open.

So, we can use linear algebra to find safe blocks to click. In addition, variables in this matrix are not as complicated as in real life since a block can be a mine (1) or empty (0). So, we can do further simplifications like this:

$$A + B + C = 3$$

$$A,B,C = 1$$

$$A + B + C = 0$$

$$A,B,C = 0$$

We can do these simplifications because a variable can be either 1 or 0. To further increase its power, we can add a row containing all unopened cells. This row must be equal to remaining bomb count.

4 - Hint Implementation

When pressed on the hint button, it sends clicked signal to gamehandler. If gamehandler is on hintmode and hintcoords is not -1, we simply call click function on the hintcoord. Else we create a linear system.

Linear system is a vector of vectors of type int. Rows of linear system are equations that has the size rows * cols + 1. Last element of this vector is the answer to this equation. Other values indicate variables. For example : button in a 10x15 grid with coordinates x = 2 and y = 3 is converted to 3 * 15 + 2 = 47. So, 47'th index of this equation must be 1.

Our algorithm first looks through the grid and tries to find opened blocks. If the block is opened and its value is not zero, we create a new vector and set its last value to blocks' value. Then we add neighboring cells to this equation. After adding all rows according to opened

blocks, we add one more equation including all unopened blocks with bomb count. This is the bomb equation.

After adding all equations to the matrix, we try to do gaussian elimination. First, we sort all equations according to their pivot point. Then we subtract upper rows from lower rows if they are necessary. By necessary, we mean all variables from the upper row must appear in the lower row if we want to subtract it. This is done to not lose information while doing row reduction. If this was a normal matrix this wouldn't be a problem, but our variables are defined in a Boolean way, so it would cause information loss.

After doing necessary subtractions, we simplify equations. First, we add rows consisting full of zeros to a set to remove them later. If we have a row of type A + B + C = 3 we remove the row and add 3 rows that indicate that these variables are mines. If we have a row of type A + B + C = 0 then we just return and make it the hint coordinate.

If we did not return after these actions, we do the same actions, but from bottom to top. And if we added a row to simplify the matrix during these eliminations, we recursively call these gaussian elimination actions again.

After these actions we try to find an equation in the matrix with answer 0. If we found a row like that, we change hintmode and hint coords and change the image of the found button.

Let's see a simple example without including bomb count :

Simple Example

After pressing the hint button, our matrix will be like this :

0 0 0 1 1 0 0 0 0 1

0 0 1 1 1 0 0 0 0 2

0 0 0 0 1 1 0 0 0 1

After sorting according to pivots :

0 0 0 1 1 1 0 0 0 2

0 0 0 1 1 0 0 0 0 1

0 0 0 0 1 1 0 0 0 1

Modified gaussian elimination from top to bottom :

0 0 1 1 1 0 0 0 0 2

0 0 0 1 1 0 0 0 0 1

0 0 0 0 1 1 0 0 0 1

There is no change because none of the lower rows have the same variables as upper rows

Modified gaussian elimination from bottom to top:

0 0 0 0 0 1 0 0 0 1

0 0 0 1 1 0 0 0 0 1

0 0 0 0 1 1 0 0 0 1

Since there is a change, we are going to do the gaussian elimination again recursively. After this call we are going to have this row :

0 0 0 1 1 0 0 0 0 1

0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0 0 1

Second row indicates that 5'th cell (x = 2,y = 1) is safe to open, so we should give it as hint.

5 - Difficulties We Faced

- Wrong parameter passing type

  To sort the matrix according to their pivots, we have used custom compareByPivot function. The header was like this:

  static bool compareByPivotPosition(std::vector<int> first, std::vector<int> second)

  Since we are passing vectors, all rows in the matrix were copied multiple times when compared. So, this has led to many crashes in our program. We solved this program by passing vectors like this:

  const std::vector<int> & first,const std::vector<int> & second

- Wrong indexing

  Instead of storing equations on a 2D vector, we have flattened it to a 1D vector. It has made our implementation easier. At first, we have used this conversion :

  X = coordinate / rows, Y = coordinate % rows

  Obviously, this conversion is wrong. We have to divide it by columns, not rows. However, we did not see any errors in early development because we have always tested on a 10x10 map, and in a 10x10 map, column count and row counts are same.

- Unnecessary row reduction

  At our first implementation our matrix reduction method was same as gaussian elimination. So whenever pivots match, we subtract rows to remove that pivot. However, minesweeper logic is not entirely compatible with standard linear algebra. In minesweeper a cell can be either 1 (bomb) or 0 (empty). Also, some equations might have connections, but we do not want to consider them to not complicate our program

  Example matrix:

1 1 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0 0

0 0 1 1 0 0 0 0 0

0 0 0 1 1 0 0 0 0

After doing normal elimination :

1 0 0 0 -1 0 0 0 0

0 1 0 0 1 0 0 0 0

0 0 1 0 -1 0 0 0 0

0 0 0 1 1 0 0 0 0

So, our first cell is depending on fifth cell. Mathematically this makes sense, however these cells can be far away and can be disconnected. So, we do not want to see calculate possibilities about far away cells.

6 – Conclusion

To conclude, we implemented a Minesweeper game with all expected functionalities and game features. We have been familiarized with C++ and QT by this project. Also, we had to find efficient algorithms while implementing some of the functions. Also, we used our knowledge from previous courses like MATH201(linear algebra) CMPE160 (Object Oriented Programming) and CMPE250 (Data Structures and Algorithms). In conclusion, we adapted the new language and submit a well-structured homework with connected to previouus information we have.